autogluon_each_location

October 9, 2023

[1]:
```
# config

label = 'y'
metric = 'mean_absolute_error'
time_limit = 60*30
presets = 'best_quality'

do_drop_ds = True

use_groups = False
n_groups = 8

auto_stack = True
num_stack_levels = 1
num_bag_folds = 0
if auto_stack:
    num_stack_levels = None
    num_bag_folds = None

use_tune_data = False
use_test_data = True
tune_and_test_length = 24*30*3 # 3 months from end, this changes the
 ↪evaluations for only test
holdout_frac = None
use_bag_holdout = False # Enable this if there is a large gap between score_val
 ↪and score_test in stack models.

sample_weight = 'sample_weight' #None
weight_evaluation = True #False
sample_weight_estimated = 2 # this changes evaluations for test and tune WTF,
 ↪cant find a fix

run_analysis = False
```

[2]:
```
import pandas as pd
import numpy as np
```

```python
import warnings
warnings.filterwarnings("ignore")

def fix_datetime(X, name):
    # Convert 'date_forecast' to datetime format and replace original column␣
↪with 'ds'
    X['ds'] = pd.to_datetime(X['date_forecast'])
    X.drop(columns=['date_forecast'], inplace=True, errors='ignore')
    X.sort_values(by='ds', inplace=True)
    X.set_index('ds', inplace=True)

    # Drop rows where the minute part of the time is not 0
    X = X[X.index.minute == 0].copy()
    return X



def convert_to_datetime(X_train_observed, X_train_estimated, X_test, y_train):
    X_train_observed = fix_datetime(X_train_observed, "X_train_observed")
    X_train_estimated = fix_datetime(X_train_estimated, "X_train_estimated")
    X_test = fix_datetime(X_test, "X_test")

    # add sample weights, which are 1 for observed and 3 for estimated
    X_train_observed["sample_weight"] = 1
    X_train_estimated["sample_weight"] = sample_weight_estimated
    X_test["sample_weight"] = sample_weight_estimated

    X_train_observed["estimated_diff_hours"] = 0
    X_train_estimated["estimated_diff_hours"] = (X_train_estimated.index - pd.
↪to_datetime(X_train_estimated["date_calc"])).dt.total_seconds() / 3600
    X_test["estimated_diff_hours"] = (X_test.index - pd.
↪to_datetime(X_test["date_calc"])).dt.total_seconds() / 3600

    X_train_estimated["estimated_diff_hours"] =␣
↪X_train_estimated["estimated_diff_hours"].astype('int64')
    # the filled once will get dropped later anyways, when we drop y nans
    X_test["estimated_diff_hours"] = X_test["estimated_diff_hours"].fillna(-50).
↪astype('int64')


    X_train_estimated.drop(columns=['date_calc'], inplace=True)
    X_test.drop(columns=['date_calc'], inplace=True)

    y_train['ds'] = pd.to_datetime(y_train['time'])
    y_train.drop(columns=['time'], inplace=True)
```

```
    y_train.sort_values(by='ds', inplace=True)
    y_train.set_index('ds', inplace=True)

    return X_train_observed, X_train_estimated, X_test, y_train




def preprocess_data(X_train_observed, X_train_estimated, X_test, y_train, location):
    # convert to datetime
    X_train_observed, X_train_estimated, X_test, y_train = convert_to_datetime(X_train_observed, X_train_estimated, X_test, y_train)

    y_train["y"] = y_train["pv_measurement"].astype('float64')
    y_train.drop(columns=['pv_measurement'], inplace=True)
    X_train = pd.concat([X_train_observed, X_train_estimated])

    # fill missng sample_weight with 3
    #X_train["sample_weight"] = X_train["sample_weight"].fillna(0)


    # clip all y values to 0 if negative
    y_train["y"] = y_train["y"].clip(lower=0)

    X_train = pd.merge(X_train, y_train, how="inner", left_index=True, right_index=True)

    # print number of nans in sample_weight
    print(f"Number of nans in sample_weight: {X_train['sample_weight'].isna().sum()}")
    # print number of nans in y
    print(f"Number of nans in y: {X_train['y'].isna().sum()}")


    X_train["location"] = location
    X_test["location"] = location

    return X_train, X_test
# Define locations
locations = ['A', 'B', 'C']

X_trains = []
X_tests = []
# Loop through locations
for loc in locations:
    print(f"Processing location {loc}...")
```

```
    # Read target training data
    y_train = pd.read_parquet(f'{loc}/train_targets.parquet')

    # Read estimated training data and add location feature
    X_train_estimated = pd.read_parquet(f'{loc}/X_train_estimated.parquet')

    # Read observed training data and add location feature
    X_train_observed= pd.read_parquet(f'{loc}/X_train_observed.parquet')

    # Read estimated test data and add location feature
    X_test_estimated = pd.read_parquet(f'{loc}/X_test_estimated.parquet')

    # Preprocess data
    X_train, X_test = preprocess_data(X_train_observed, X_train_estimated,␣
↪X_test_estimated, y_train, loc)

    X_trains.append(X_train)
    X_tests.append(X_test)

# Concatenate all data and save to csv
X_train = pd.concat(X_trains)
X_test = pd.concat(X_tests)
```

```
Processing location A…
Number of nans in sample_weight: 0
Number of nans in y: 0
Processing location B…
Number of nans in sample_weight: 0
Number of nans in y: 4
Processing location C…
Number of nans in sample_weight: 0
Number of nans in y: 6059
```

1 Feature enginering

[3]:
```
import numpy as np
import pandas as pd

X_train.dropna(subset=['y'], inplace=True)

if not do_drop_ds:
    # add hour datetime feature
    X_train["hour"] = X_train.index.hour
    X_test["hour"] = X_test.index.hour

#print(X_train.head())
```

```
if use_groups:
    # fix groups for cross validation
    locations = X_train['location'].unique()  # Assuming 'location' is the name␣
↪of the column representing locations

    grouped_dfs = []  # To store data frames split by location

    # Loop through each unique location
    for loc in locations:
        loc_df = X_train[X_train['location'] == loc]

        # Sort the DataFrame for this location by the time column
        loc_df = loc_df.sort_index()

        # Calculate the size of each group for this location
        group_size = len(loc_df) // n_groups

        # Create a new 'group' column for this location
        loc_df['group'] = np.repeat(range(n_groups),␣
↪repeats=[group_size]*(n_groups-1) + [len(loc_df) - group_size*(n_groups-1)])

        # Append to list of grouped DataFrames
        grouped_dfs.append(loc_df)

    # Concatenate all the grouped DataFrames back together
    X_train = pd.concat(grouped_dfs)
    X_train.sort_index(inplace=True)
    print(X_train["group"].head())





to_drop = ["snow_drift:idx", "snow_density:kgm3"]

X_train.drop(columns=to_drop, inplace=True)
X_test.drop(columns=to_drop, inplace=True)

X_train.to_csv('X_train_raw.csv', index=True)
X_test.to_csv('X_test_raw.csv', index=True)
```

[4]:
```
from autogluon.tabular import TabularDataset, TabularPredictor
from autogluon.timeseries import TimeSeriesDataFrame
import numpy as np
train_data = TabularDataset('X_train_raw.csv')
```

```python
# set group column of train_data be increasing from 0 to 7 based on time, the␣
↪first 1/8 of the data is group 0, the second 1/8 of the data is group 1, etc.
train_data['ds'] = pd.to_datetime(train_data['ds'])
train_data = train_data.sort_values(by='ds')

# # print size of the group for each location
# for loc in locations:
#     print(f"Location {loc}:")
#     print(train_data[train_data["location"] == loc].groupby('group').size())


# get end date of train data and subtract 3 months
split_time = pd.to_datetime(train_data["ds"]).max() - pd.
↪Timedelta(hours=tune_and_test_length)
train_set = TabularDataset(train_data[train_data["ds"] < split_time])
test_set = TabularDataset(train_data[train_data["ds"] >= split_time])
if use_groups:
    test_set = test_set.drop(columns=['group'])

if do_drop_ds:
    train_set = train_set.drop(columns=['ds'])
    test_set = test_set.drop(columns=['ds'])
    train_data = train_data.drop(columns=['ds'])


def normalize_sample_weights_per_location(df):
    for loc in locations:
        loc_df = df[df["location"] == loc]
        loc_df["sample_weight"] = loc_df["sample_weight"] /␣
↪loc_df["sample_weight"].sum() * loc_df.shape[0]
        df[df["location"] == loc] = loc_df
    return df

tuning_data = None
if use_tune_data:
    train_data = train_set
    if use_test_data:
        # split test_set in half, use first half for tuning
        tuning_data, test_data = [], []
        for loc in locations:
            loc_test_set = test_set[test_set["location"] == loc]
            loc_tuning_data = loc_test_set.iloc[:len(loc_test_set)//2]
            loc_test_data = loc_test_set.iloc[len(loc_test_set)//2:]
            tuning_data.append(loc_tuning_data)
            test_data.append(loc_test_data)
        tuning_data = pd.concat(tuning_data)
        test_data = pd.concat(test_data)
```

```
        print("Shapes of tuning and test", tuning_data.shape[0], test_data.
 ↪shape[0], tuning_data.shape[0] + test_data.shape[0])

    else:
        tuning_data = test_set
        print("Shape of tuning", tuning_data.shape[0])

    # ensure sample weights for your tuning data sum to the number of rows in␣
 ↪the tuning data.
    tuning_data = normalize_sample_weights_per_location(tuning_data)


else:
    if use_test_data:
        train_data = train_set
        test_data = test_set
        print("Shape of test", test_data.shape[0])

# ensure sample weights for your training (or tuning) data sum to the number of␣
 ↪rows in the training (or tuning) data.
train_data = normalize_sample_weights_per_location(train_data)
if use_test_data:
    test_data = normalize_sample_weights_per_location(test_data)
```

```
Shape of test 5791
```

[5]:
```
if run_analysis:
    import autogluon.eda.auto as auto
    auto.dataset_overview(train_data=train_data, test_data=test_data,␣
 ↪label="y", sample=None)
```

[6]:
```
if run_analysis:
    auto.target_analysis(train_data=train_data, label="y")
```

2 Starting

[7]:
```
import os



# Get the last submission number
last_submission_number = int(max([int(filename.split('_')[1].split('.')[0]) for␣
 ↪filename in os.listdir('submissions') if "submission" in filename]))
print("Last submission number:", last_submission_number)
print("Now creating submission number:", last_submission_number + 1)

# Create the new filename
new_filename = f'submission_{last_submission_number + 1}'
```

```python
hello = os.environ.get('HELLO')
if hello is not None:
    new_filename += f'_{hello}'

print("New filename:", new_filename)
```

```
Last submission number: 83
Now creating submission number: 84
New filename: submission_84
```

[8]:
```python
predictors = [None, None, None]
```

[9]:
```python
def fit_predictor_for_location(loc):
    print(f"Training model for location {loc}...")
    # sum of sample weights for this location, and number of rows, for both␣
 ↪train and tune data and test data
    print("Train data sample weight sum:", train_data[train_data["location"] ==␣
 ↪loc]["sample_weight"].sum())
    print("Train data number of rows:", train_data[train_data["location"] ==␣
 ↪loc].shape[0])
    if use_tune_data:
        print("Tune data sample weight sum:",␣
 ↪tuning_data[tuning_data["location"] == loc]["sample_weight"].sum())
        print("Tune data number of rows:", tuning_data[tuning_data["location"]␣
 ↪== loc].shape[0])
    if use_test_data:
        print("Test data sample weight sum:", test_data[test_data["location"]␣
 ↪== loc]["sample_weight"].sum())
        print("Test data number of rows:", test_data[test_data["location"] ==␣
 ↪loc].shape[0])
    predictor = TabularPredictor(
        label=label,
        eval_metric=metric,
        path=f"AutogluonModels/{new_filename}_{loc}",
        sample_weight=sample_weight,
        weight_evaluation=weight_evaluation,
        groups="group" if use_groups else None,
    ).fit(
        train_data=train_data[train_data["location"] == loc],
        time_limit=time_limit,
        #presets=presets,
        num_stack_levels=num_stack_levels,
        num_bag_folds=num_bag_folds if not use_groups else 2,# just put␣
 ↪somethin, will be overwritten anyways
        tuning_data=tuning_data[tuning_data["location"] == loc] if␣
 ↪use_tune_data else None,
```

```python
        use_bag_holdout=use_bag_holdout,
        holdout_frac=holdout_frac,
    )

    # evaluate on test data
    if use_test_data:
        # drop sample_weight column
        t = test_data[test_data["location"] == loc]#.
↪drop(columns=["sample_weight"])
        perf = predictor.evaluate(t)
        print("Evaluation on test data:")
        print(perf[predictor.eval_metric.name])

    return predictor

loc = "A"
predictors[0] = fit_predictor_for_location(loc)
```

```
Training model for location A…
Train data sample weight sum: 31900.000000000007
Train data number of rows: 31900
Test data sample weight sum: 2161

Values in column 'sample_weight' used as sample weights instead of predictive
features. Evaluation will report weighted metrics, so ensure same column exists
in test data.
Beginning AutoGluon training … Time limit = 1800s
AutoGluon will save models to "AutogluonModels/submission_84_A/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   311.54 GB / 315.93 GB (98.6%)
Train Data Rows:    31900
Train Data Columns: 46
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (5733.42, 0.0, 633.132, 1165.64686)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    132415.1 MB
        Train Data (Original)  Memory Usage: 13.08 MB (0.0% of available memory)
```

```
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 3 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
```

```
Test data number of rows: 2161
```

```
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 42 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])   :  1 | ['estimated_diff_hours']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 39 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])       :  1 | ['estimated_diff_hours']
                ('int', ['bool']) :  3 | ['is_day:idx', 'is_in_shadow:idx',
'wind_speed_w_1000hPa:ms']
        0.2s = Fit runtime
        43 features in original data used to generate 43 features in processed
data.
        Train Data (Processed) Memory Usage: 10.3 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.19s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
Automatically generating train/validation split with
holdout_frac=0.07836990595611286, Train Rows: 29400, Val Rows: 2500
User-specified model hyperparameters to be fit:
{
```

```
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
Fitting model: KNeighborsUnif … Training model for up to 1799.81s of the
1799.8s of remaining time.
        -269.8458        = Validation score   (-mean_absolute_error)
        0.04s    = Training   runtime
        0.07s    = Validation runtime
Fitting model: KNeighborsDist … Training model for up to 1799.69s of the
1799.69s of remaining time.
        -272.3871        = Validation score   (-mean_absolute_error)
        0.04s    = Training   runtime
        0.04s    = Validation runtime
Fitting model: LightGBMXT … Training model for up to 1799.6s of the 1799.59s
of remaining time.

[1000]  valid_set's l1: 170.878
[2000]  valid_set's l1: 166.132
[3000]  valid_set's l1: 163.488
[4000]  valid_set's l1: 162.078
[5000]  valid_set's l1: 161.461
[6000]  valid_set's l1: 160.72
[7000]  valid_set's l1: 160.253
[8000]  valid_set's l1: 159.878
[9000]  valid_set's l1: 159.389
[10000] valid_set's l1: 159.205

        -159.1975        = Validation score   (-mean_absolute_error)
        14.12s   = Training   runtime
        0.17s    = Validation runtime
Fitting model: LightGBM … Training model for up to 1785.01s of the 1785.0s of
remaining time.
```

```
[1000]  valid_set's l1: 172.354
[2000]  valid_set's l1: 169.481
[3000]  valid_set's l1: 168.697
[4000]  valid_set's l1: 168.103
[5000]  valid_set's l1: 168.155
        -168.0752        = Validation score   (-mean_absolute_error)
        7.97s    = Training   runtime
        0.07s    = Validation runtime
Fitting model: RandomForestMSE … Training model for up to 1776.82s of the
1776.81s of remaining time.
        -176.8389        = Validation score   (-mean_absolute_error)
        8.06s    = Training   runtime
        0.09s    = Validation runtime
Fitting model: CatBoost … Training model for up to 1768.21s of the 1768.21s of
remaining time.
        -171.5531        = Validation score   (-mean_absolute_error)
        119.15s  = Training   runtime
        0.01s    = Validation runtime
Fitting model: ExtraTreesMSE … Training model for up to 1649.02s of the
1649.01s of remaining time.
        -176.3634        = Validation score   (-mean_absolute_error)
        1.76s    = Training   runtime
        0.08s    = Validation runtime
Fitting model: NeuralNetFastAI … Training model for up to 1646.68s of the
1646.67s of remaining time.
        -182.3609        = Validation score   (-mean_absolute_error)
        27.28s   = Training   runtime
        0.04s    = Validation runtime
Fitting model: XGBoost … Training model for up to 1619.33s of the 1619.32s of
remaining time.
        -176.8785        = Validation score   (-mean_absolute_error)
        1.29s    = Training   runtime
        0.01s    = Validation runtime
Fitting model: NeuralNetTorch … Training model for up to 1618.01s of the
1618.0s of remaining time.
        -166.1122        = Validation score   (-mean_absolute_error)
        52.16s   = Training   runtime
        0.04s    = Validation runtime
Fitting model: LightGBMLarge … Training model for up to 1565.8s of the 1565.8s
of remaining time.
[1000]  valid_set's l1: 162.299
[2000]  valid_set's l1: 160.562
[3000]  valid_set's l1: 160.294
[4000]  valid_set's l1: 160.138
[5000]  valid_set's l1: 160.091
[6000]  valid_set's l1: 160.08
[7000]  valid_set's l1: 160.069
```

```
[8000]  valid_set's l1: 160.065
[9000]  valid_set's l1: 160.064
[10000] valid_set's l1: 160.063

        -160.0634        = Validation score   (-mean_absolute_error)
        48.26s    = Training   runtime
        0.29s     = Validation runtime
Fitting model: WeightedEnsemble_L2 … Training model for up to 360.0s of the
1516.04s of remaining time.
        -153.4447        = Validation score   (-mean_absolute_error)
        0.46s     = Training   runtime
        0.0s      = Validation runtime
AutoGluon training complete, total runtime = 284.47s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_84_A/")
WARNING: eval_metric='pearsonr' does not support sample weights so they will be
ignored in reported metric.
Evaluation: mean_absolute_error on test data: -190.03460684764755
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -190.03460684764755,
    "root_mean_squared_error": -417.1069097664702,
    "mean_squared_error": -173978.17417493433,
    "r2": 0.873767738357695,
    "pearsonr": 0.9351134793298022,
    "median_absolute_error": -11.291558265686035
}

Evaluation on test data:
-190.03460684764755
```

[10]:
```
loc = "B"
predictors[1] = fit_predictor_for_location(loc)
```

```
Values in column 'sample_weight' used as sample weights instead of predictive
features. Evaluation will report weighted metrics, so ensure same column exists
in test data.
Beginning AutoGluon training … Time limit = 1800s
AutoGluon will save models to "AutogluonModels/submission_84_B/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   310.55 GB / 315.93 GB (98.3%)
Train Data Rows:    30768
```

```
Train Data Columns: 46
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (1152.3, -0.0, 97.74541, 195.0957)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    130666.32 MB
        Train Data (Original)  Memory Usage: 12.62 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 3 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
```

```
Training model for location B…
Train data sample weight sum: 30767.999999999993
Train data number of rows: 30768
Test data sample weight sum: 2051
Test data number of rows: 2051
```

```
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 42 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])   :  1 | ['estimated_diff_hours']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 39 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
```

```
                ('int', [])       :  1 | ['estimated_diff_hours']
                ('int', ['bool']) :  3 | ['is_day:idx', 'is_in_shadow:idx',
'wind_speed_w_1000hPa:ms']
        0.2s = Fit runtime
        43 features in original data used to generate 43 features in processed
data.
        Train Data (Processed) Memory Usage: 9.94 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.18s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
Automatically generating train/validation split with
holdout_frac=0.0812532501300052, Train Rows: 28268, Val Rows: 2500
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
Fitting model: KNeighborsUnif … Training model for up to 1799.82s of the
1799.81s of remaining time.
        -55.194  = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.04s    = Validation runtime
Fitting model: KNeighborsDist … Training model for up to 1799.74s of the
1799.73s of remaining time.
        -55.0011         = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.04s    = Validation runtime
Fitting model: LightGBMXT … Training model for up to 1799.65s of the 1799.65s
```

```
of remaining time.
[1000]  valid_set's l1: 34.1153
[2000]  valid_set's l1: 31.9514
[3000]  valid_set's l1: 30.9565
[4000]  valid_set's l1: 30.2929
[5000]  valid_set's l1: 29.8272
[6000]  valid_set's l1: 29.4326
[7000]  valid_set's l1: 29.1915
[8000]  valid_set's l1: 28.9733
[9000]  valid_set's l1: 28.8002
[10000] valid_set's l1: 28.6176
        -28.6164         = Validation score   (-mean_absolute_error)
        13.99s   = Training   runtime
        0.19s    = Validation runtime
Fitting model: LightGBM … Training model for up to 1785.18s of the 1785.18s of
remaining time.
[1000]  valid_set's l1: 32.1069
[2000]  valid_set's l1: 30.6848
[3000]  valid_set's l1: 30.0177
[4000]  valid_set's l1: 29.5944
[5000]  valid_set's l1: 29.3602
[6000]  valid_set's l1: 29.2298
[7000]  valid_set's l1: 29.1699
[8000]  valid_set's l1: 29.1357
[9000]  valid_set's l1: 29.0937
[10000] valid_set's l1: 29.0594
        -29.0594         = Validation score   (-mean_absolute_error)
        14.68s   = Training   runtime
        0.18s    = Validation runtime
Fitting model: RandomForestMSE … Training model for up to 1770.0s of the
1769.99s of remaining time.
        -34.0014         = Validation score   (-mean_absolute_error)
        9.19s    = Training   runtime
        0.09s    = Validation runtime
Fitting model: CatBoost … Training model for up to 1760.35s of the 1760.35s of
remaining time.
        -31.1926         = Validation score   (-mean_absolute_error)
        119.06s  = Training   runtime
        0.01s    = Validation runtime
Fitting model: ExtraTreesMSE … Training model for up to 1641.25s of the
1641.24s of remaining time.
        -34.9603         = Validation score   (-mean_absolute_error)
        1.96s    = Training   runtime
        0.09s    = Validation runtime
Fitting model: NeuralNetFastAI … Training model for up to 1638.79s of the
1638.78s of remaining time.
```

```
        -38.9413         = Validation score   (-mean_absolute_error)
        26.25s   = Training   runtime
        0.04s    = Validation runtime
Fitting model: XGBoost … Training model for up to 1612.47s of the 1612.46s of
remaining time.
        -32.361  = Validation score   (-mean_absolute_error)
        24.2s    = Training   runtime
        0.21s    = Validation runtime
Fitting model: NeuralNetTorch … Training model for up to 1587.9s of the
1587.9s of remaining time.
        -32.7404         = Validation score   (-mean_absolute_error)
        100.54s  = Training   runtime
        0.04s    = Validation runtime
Fitting model: LightGBMLarge … Training model for up to 1487.32s of the
1487.31s of remaining time.

[1000]  valid_set's l1: 29.0886
[2000]  valid_set's l1: 28.303
[3000]  valid_set's l1: 28.1063
[4000]  valid_set's l1: 28.0366
[5000]  valid_set's l1: 28.0061
[6000]  valid_set's l1: 27.9919
[7000]  valid_set's l1: 27.9858
[8000]  valid_set's l1: 27.9833
[9000]  valid_set's l1: 27.9823
[10000] valid_set's l1: 27.9816

        -27.9816         = Validation score   (-mean_absolute_error)
        47.27s   = Training   runtime
        0.31s    = Validation runtime
Fitting model: WeightedEnsemble_L2 … Training model for up to 360.0s of the
1438.44s of remaining time.
        -27.3393         = Validation score   (-mean_absolute_error)
        0.44s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 362.04s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_84_B/")
WARNING: eval_metric='pearsonr' does not support sample weights so they will be
ignored in reported metric.
Evaluation: mean_absolute_error on test data: -37.780304819862444
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -37.780304819862444,
    "root_mean_squared_error": -82.57371750173203,
    "mean_squared_error": -6818.418822055848,
```

```
    "r2": 0.7806994305325522,
    "pearsonr": 0.9090115264745972,
    "median_absolute_error": -8.113659752314257
}
```

```
Evaluation on test data:
-37.780304819862444
```

[11]:
```python
loc = "C"
predictors[2] = fit_predictor_for_location(loc)
```

```
Values in column 'sample_weight' used as sample weights instead of predictive
features. Evaluation will report weighted metrics, so ensure same column exists
in test data.
Beginning AutoGluon training … Time limit = 1800s
AutoGluon will save models to "AutogluonModels/submission_84_C/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   309.61 GB / 315.93 GB (98.0%)
Train Data Rows:    24492
Train Data Columns: 46
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and label-values can't be converted to int).
        Label info (max, min, mean, stddev): (999.6, 0.0, 78.11911, 167.50151)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    130436.68 MB
        Train Data (Original)  Memory Usage: 10.04 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 2 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
```

```
		Fitting DropDuplicatesFeatureGenerator…
	Useless Original Features (Count: 2): ['elevation:m', 'location']
		These features carry no predictive signal and should be manually
investigated.
		This is typically a feature which has the same value for all
rows.
		These features do not need to be present at inference time.
	Types of features in original data (raw dtype, special dtypes):
		('float', []) : 42 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
```

```
Training model for location C…
Train data sample weight sum: 24492.000000000007
Train data number of rows: 24492
Test data sample weight sum: 1579
Test data number of rows: 1579
```

```
		('int', [])   :  1 | ['estimated_diff_hours']
	Types of features in processed data (raw dtype, special dtypes):
		('float', [])     : 40 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
		('int', [])       :  1 | ['estimated_diff_hours']
		('int', ['bool']) :  2 | ['is_day:idx', 'is_in_shadow:idx']
	0.1s = Fit runtime
	43 features in original data used to generate 43 features in processed
data.
	Train Data (Processed) Memory Usage: 8.08 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.16s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
	This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
	To change this, specify the eval_metric parameter of Predictor()
Automatically generating train/validation split with holdout_frac=0.1, Train
Rows: 22042, Val Rows: 2450
User-specified model hyperparameters to be fit:
{
	'NN_TORCH': {},
	'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
	'CAT': {},
	'XGB': {},
	'FASTAI': {},
	'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
```

```
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
Fitting model: KNeighborsUnif … Training model for up to 1799.84s of the
1799.83s of remaining time.
        -31.566  = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.03s    = Validation runtime
Fitting model: KNeighborsDist … Training model for up to 1799.76s of the
1799.76s of remaining time.
        -31.6219         = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.03s    = Validation runtime
Fitting model: LightGBMXT … Training model for up to 1799.69s of the 1799.69s
of remaining time.

[1000]  valid_set's l1: 18.2091
[2000]  valid_set's l1: 17.6836
[3000]  valid_set's l1: 17.4143
[4000]  valid_set's l1: 17.2751
[5000]  valid_set's l1: 17.1821
[6000]  valid_set's l1: 17.1353
[7000]  valid_set's l1: 17.1077
[8000]  valid_set's l1: 17.0671
[9000]  valid_set's l1: 17.0307
[10000] valid_set's l1: 17.0159

        -17.0155         = Validation score   (-mean_absolute_error)
        13.26s   = Training   runtime
        0.18s    = Validation runtime
Fitting model: LightGBM … Training model for up to 1785.97s of the 1785.97s of
remaining time.

[1000]  valid_set's l1: 18.2564
[2000]  valid_set's l1: 18.0057
[3000]  valid_set's l1: 17.9332
[4000]  valid_set's l1: 17.9
[5000]  valid_set's l1: 17.8853
[6000]  valid_set's l1: 17.8762
[7000]  valid_set's l1: 17.8736
[8000]  valid_set's l1: 17.8687
[9000]  valid_set's l1: 17.8658
```

```
[10000] valid_set's l1: 17.863
        -17.8628         = Validation score   (-mean_absolute_error)
        14.01s   = Training   runtime
        0.18s    = Validation runtime
Fitting model: RandomForestMSE … Training model for up to 1771.53s of the
1771.52s of remaining time.
        -19.2476         = Validation score   (-mean_absolute_error)
        4.85s    = Training   runtime
        0.09s    = Validation runtime
Fitting model: CatBoost … Training model for up to 1766.43s of the 1766.42s of
remaining time.
        -17.9528         = Validation score   (-mean_absolute_error)
        118.46s  = Training   runtime
        0.01s    = Validation runtime
Fitting model: ExtraTreesMSE … Training model for up to 1647.93s of the
1647.92s of remaining time.
        -19.1696         = Validation score   (-mean_absolute_error)
        1.1s     = Training   runtime
        0.09s    = Validation runtime
Fitting model: NeuralNetFastAI … Training model for up to 1646.56s of the
1646.55s of remaining time.
        -19.3985         = Validation score   (-mean_absolute_error)
        21.46s   = Training   runtime
        0.04s    = Validation runtime
Fitting model: XGBoost … Training model for up to 1625.03s of the 1625.03s of
remaining time.
        -18.004  = Validation score   (-mean_absolute_error)
        23.66s   = Training   runtime
        0.2s     = Validation runtime
Fitting model: NeuralNetTorch … Training model for up to 1601.02s of the
1601.02s of remaining time.
        -18.0551         = Validation score   (-mean_absolute_error)
        79.74s   = Training   runtime
        0.04s    = Validation runtime
Fitting model: LightGBMLarge … Training model for up to 1521.24s of the
1521.24s of remaining time.

[1000]  valid_set's l1: 17.2394
[2000]  valid_set's l1: 17.1378
[3000]  valid_set's l1: 17.1097
[4000]  valid_set's l1: 17.1042
[5000]  valid_set's l1: 17.103
[6000]  valid_set's l1: 17.1025
[7000]  valid_set's l1: 17.1023
[8000]  valid_set's l1: 17.1022
[9000]  valid_set's l1: 17.1021
[10000] valid_set's l1: 17.1021
```

```
        -17.1021         = Validation score   (-mean_absolute_error)
        46.6s    = Training   runtime
        0.36s    = Validation runtime
Fitting model: WeightedEnsemble_L2 … Training model for up to 360.0s of the
1472.99s of remaining time.
        -16.2378         = Validation score   (-mean_absolute_error)
        0.44s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 327.49s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_84_C/")
WARNING: eval_metric='pearsonr' does not support sample weights so they will be
ignored in reported metric.
Evaluation: mean_absolute_error on test data: -30.771432015969143
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -30.771432015969143,
    "root_mean_squared_error": -63.5906665887193,
    "mean_squared_error": -4043.7728771976617,
    "r2": 0.7879842629329407,
    "pearsonr": 0.8936165759890148,
    "median_absolute_error": -3.2815890502929648
}
```

```
Evaluation on test data:
-30.771432015969143
```

3 Submit

[12]:
```python
import pandas as pd
import matplotlib.pyplot as plt

train_data_with_dates = TabularDataset('X_train_raw.csv')
train_data_with_dates["ds"] = pd.to_datetime(train_data_with_dates["ds"])

test_data = TabularDataset('X_test_raw.csv')
test_data["ds"] = pd.to_datetime(test_data["ds"])
#test_data
```

```
Loaded data from: X_train_raw.csv | Columns = 48 / 48 | Rows = 92951 -> 92951
Loaded data from: X_test_raw.csv | Columns = 47 / 47 | Rows = 2160 -> 2160
```

[13]:
```python
test_ids = TabularDataset('test.csv')
test_ids["time"] = pd.to_datetime(test_ids["time"])
# merge test_data with test_ids
```

```
test_data_merged = pd.merge(test_data, test_ids, how="inner", right_on=["time",␣
↪"location"], left_on=["ds", "location"])

#test_data_merged
```

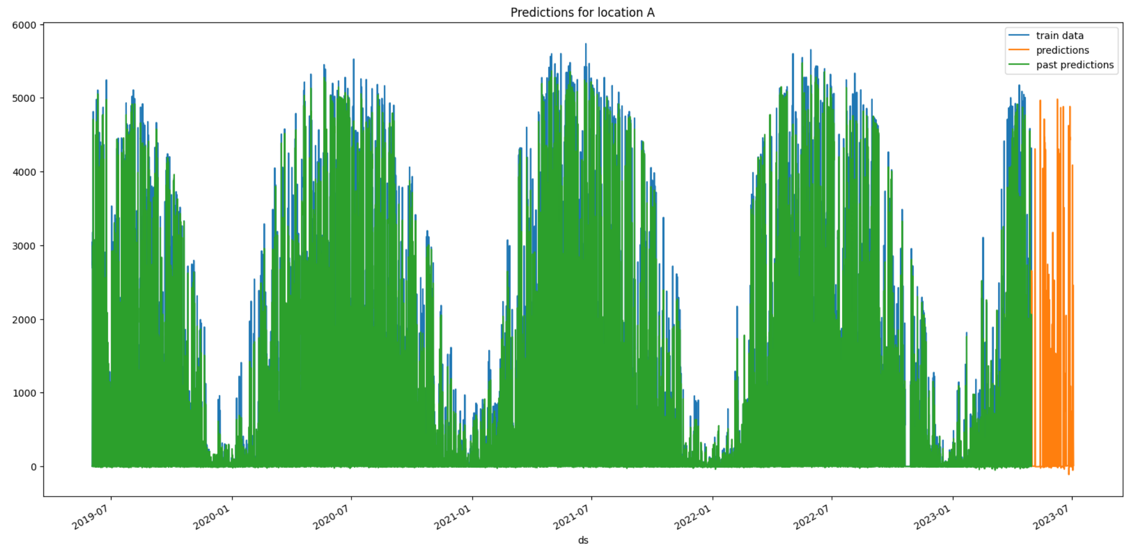
```
Loaded data from: test.csv | Columns = 4 / 4 | Rows = 2160 -> 2160
```

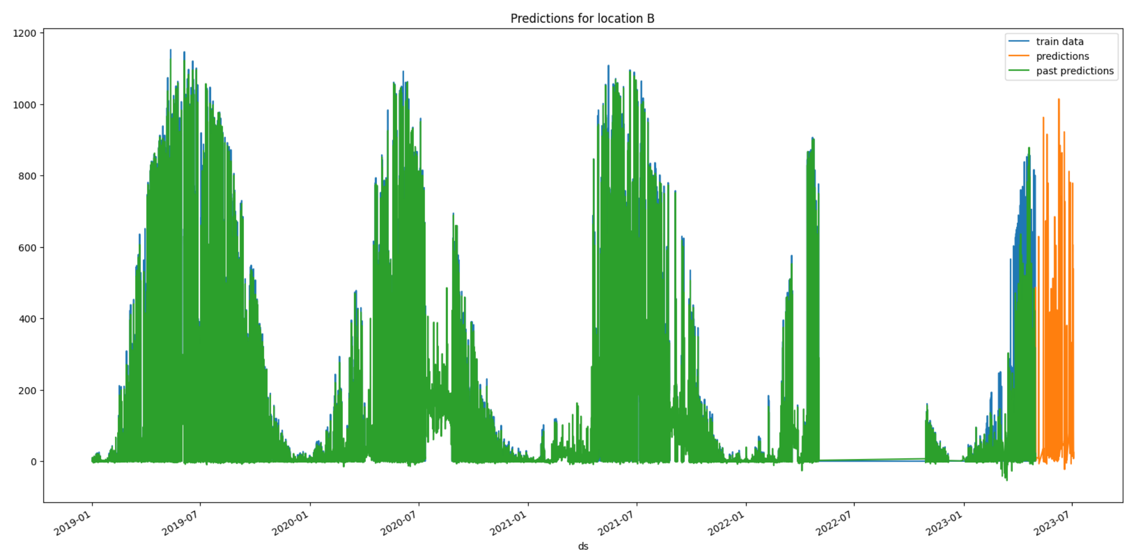
[14]:

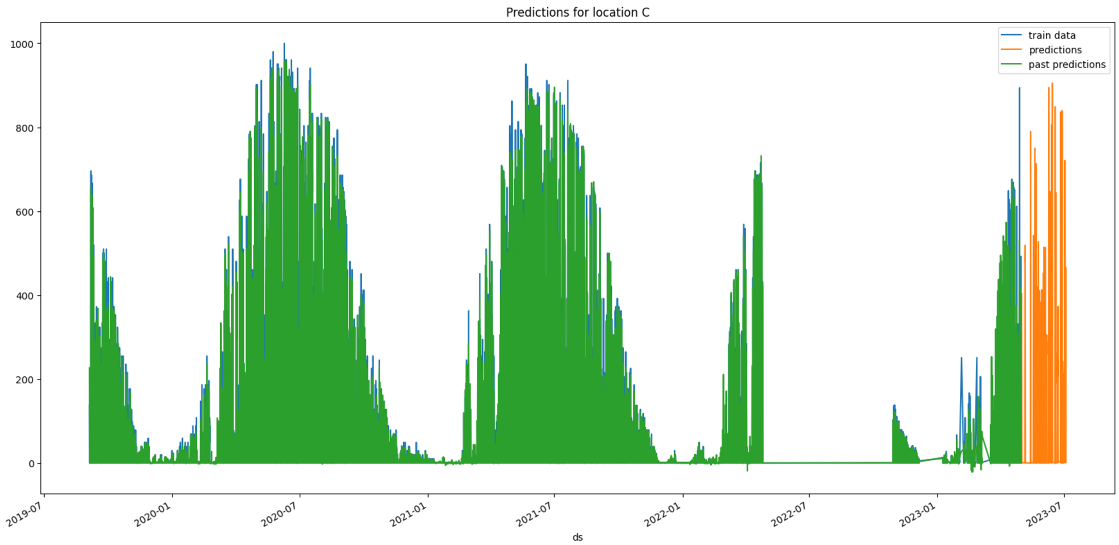
```
# predict, grouped by location
predictions = []
location_map = {
    "A": 0,
    "B": 1,
    "C": 2
}
for loc, group in test_data.groupby('location'):
    i = location_map[loc]
    subset = test_data_merged[test_data_merged["location"] == loc].
↪reset_index(drop=True)
    #print(subset)
    pred = predictors[i].predict(subset)
    subset["prediction"] = pred
    predictions.append(subset)

    # get past predictions
    past_pred = predictors[i].
↪predict(train_data_with_dates[train_data_with_dates["location"] == loc])
    train_data_with_dates.loc[train_data_with_dates["location"] == loc,␣
↪"prediction"] = past_pred
```

[15]:

```
# plot predictions for location A, in addition to train data for A
for loc, idx in location_map.items():
    fig, ax = plt.subplots(figsize=(20, 10))
    # plot train data
    train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',␣
↪y='y', ax=ax, label="train data")

    # plot predictions
    predictions[idx].plot(x='ds', y='prediction', ax=ax, label="predictions")

    # plot past predictions
    train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',␣
↪y='prediction', ax=ax, label="past predictions")

    # title
    ax.set_title(f"Predictions for location {loc}")
```


Predictions for location A
train data
predictions
past predictions
6000
5000
4000
3000
2000
1000
0
2019-07
2020-01
2020-07
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07
ds


Predictions for location B
train data
predictions
past predictions
1200
1000
800
600
400
200
0
2019-01
2019-07
2020-01
2020-07
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07
ds



[16]:
```
# concatenate predictions
submissions_df = pd.concat(predictions)
submissions_df = submissions_df[["id", "prediction"]]
submissions_df
```

[16]:
```
       id  prediction
0       0   -1.870822
1       1   -1.611267
2       2    1.663182
3       3   42.452042
4       4  344.464111
..    ...         ...
715  2155   55.469818
716  2156   35.071697
717  2157    9.472363
718  2158    1.779894
719  2159    1.641781

[2160 rows x 2 columns]
```

[17]:
```
# Save the submission DataFrame to submissions folder, create new name based on
 ↪last submission, format is submission_<last_submission_number + 1>.csv

# Save the submission
print(f"Saving submission to submissions/{new_filename}.csv")
submissions_df.to_csv(os.path.join('submissions', f"{new_filename}.csv"),
 ↪index=False)
print("jall1a")
```

```
Saving submission to submissions/submission_84.csv
jall1a
```

[18]:
```python
# save this running notebook
from IPython.display import display, Javascript
import time

# hei123

display(Javascript("IPython.notebook.save_checkpoint();"))

time.sleep(3)
```

```
<IPython.core.display.Javascript object>
```

[19]:
```python
# save this notebook to submissions folder
import subprocess
import os
subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
 ↪join('notebook_pdfs', f"hei.pdf"), "autogluon_each_location.ipynb"])
```

```
[NbConvertApp] Converting notebook autogluon_each_location.ipynb to pdf
/opt/conda/lib/python3.10/site-packages/nbconvert/utils/pandoc.py:51:
RuntimeWarning: You are using an unsupported version of pandoc (2.9.2.1).
Your version must be at least (2.14.2) but less than (4.0.0).
Refer to https://pandoc.org/installing.html.
Continuing with doubts…
  check_pandoc_version()
[NbConvertApp] Writing 110473 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 89880 bytes to notebook_pdfs/hei.pdf
```

[19]:
```
CompletedProcess(args=['jupyter', 'nbconvert', '--to', 'pdf', '--output',
'notebook_pdfs/hei.pdf', 'autogluon_each_location.ipynb'], returncode=0)
```

[20]:
```python
# feature importance
location="A"
split_time = pd.Timestamp("2022-10-28 22:00:00")
estimated = train_data_with_dates[train_data_with_dates["ds"] >= split_time]
estimated = estimated[estimated["location"] == location]
predictors[0].feature_importance(feature_stage="original", data=estimated,␣
 ↪time_limit=60*10)
```

```
These features in provided data are not utilized by the predictor and will be
```

```
ignored: ['ds', 'elevation:m', 'sample_weight', 'location', 'prediction']
Computing feature importance via permutation shuffling for 43 features using
4394 rows with 10 shuffle sets… Time limit: 600s…
        623.92s = Expected runtime (62.39s per shuffle set)
        351.84s = Actual runtime (Completed 10 of 10 shuffle sets)
```

[20]:

	importance	stddev	p_value	n \
direct_rad:W	147.254592	1.840335	5.984714e-19	10
clear_sky_rad:W	93.660755	1.738334	2.101212e-17	10
diffuse_rad:W	70.030272	2.030611	1.162961e-15	10
sun_azimuth:d	59.041930	3.065140	2.183457e-13	10
sun_elevation:d	31.723252	0.881036	7.891388e-16	10
direct_rad_1h:J	31.618525	0.746865	1.839313e-16	10
clear_sky_energy_1h:J	28.398144	1.222731	4.066758e-14	10
diffuse_rad_1h:J	16.209468	1.117848	2.791123e-12	10
effective_cloud_cover:p	14.524108	0.922336	1.332301e-12	10
total_cloud_cover:p	14.274103	0.835227	6.392552e-13	10
wind_speed_u_10m:ms	9.529033	1.081217	2.394135e-10	10
cloud_base_agl:m	7.416030	0.501223	2.329635e-12	10
snow_water:kgm2	6.631639	0.793626	3.846266e-10	10
is_day:idx	5.909227	0.305931	2.130087e-13	10
fresh_snow_24h:cm	5.308035	0.586957	1.903027e-10	10
msl_pressure:hPa	5.146734	0.482408	4.353021e-11	10
relative_humidity_1000hPa:p	5.008415	0.665083	9.692652e-10	10
visibility:m	4.859585	0.461801	4.922165e-11	10
wind_speed_10m:ms	4.671583	0.703471	2.951091e-09	10
sfc_pressure:hPa	4.604262	0.755454	6.291200e-09	10
ceiling_height_agl:m	4.409944	0.430182	6.218205e-11	10
pressure_50m:hPa	4.030376	0.599064	2.630290e-09	10
wind_speed_v_10m:ms	3.955984	0.783451	3.273485e-08	10
is_in_shadow:idx	3.920461	0.194659	1.463312e-13	10
pressure_100m:hPa	2.832170	0.503508	1.273445e-08	10
t_1000hPa:K	2.130411	1.164024	1.317638e-04	10
air_density_2m:kgm3	1.905312	0.736314	9.235284e-06	10
fresh_snow_6h:cm	1.800157	0.249171	1.400076e-09	10
fresh_snow_12h:cm	1.742519	0.319152	1.653365e-08	10
estimated_diff_hours	1.645522	0.212687	7.625252e-10	10
snow_depth:cm	1.545368	0.396405	3.058813e-07	10
super_cooled_liquid_water:kgm2	1.420476	0.400413	6.819216e-07	10
fresh_snow_3h:cm	0.968096	0.219209	1.047428e-07	10
precip_5min:mm	0.895930	0.444144	6.420545e-05	10
dew_point_2m:K	0.699648	0.457302	4.617174e-04	10
dew_or_rime:idx	0.602567	0.209757	3.955448e-06	10
precip_type_5min:idx	0.482961	0.267444	1.451975e-04	10
fresh_snow_1h:cm	0.470631	0.212665	3.168775e-05	10
absolute_humidity_2m:gm3	0.241512	0.163638	5.866296e-04	10
rain_water:kgm2	0.197383	0.125546	3.840384e-04	10

prob_rime:p	0.102458	0.158544	3.567423e-02	10
wind_speed_w_1000hPa:ms	0.000000	0.000000	5.000000e-01	10
snow_melt_10min:mm	-0.094976	0.134779	9.735819e-01	10

	p99_high	p99_low
direct_rad:W	149.145883	145.363302
clear_sky_rad:W	95.447220	91.874290
diffuse_rad:W	72.117107	67.943437
sun_azimuth:d	62.191938	55.891921
sun_elevation:d	32.628683	30.817821
direct_rad_1h:J	32.386070	30.850981
clear_sky_energy_1h:J	29.654731	27.141558
diffuse_rad_1h:J	17.358267	15.060668
effective_cloud_cover:p	15.471982	13.576234
total_cloud_cover:p	15.132456	13.415750
wind_speed_u_10m:ms	10.640187	8.417879
cloud_base_agl:m	7.931132	6.900929
snow_water:kgm2	7.447239	5.816039
is_day:idx	6.223629	5.594825
fresh_snow_24h:cm	5.911243	4.704826
msl_pressure:hPa	5.642498	4.650969
relative_humidity_1000hPa:p	5.691913	4.324917
visibility:m	5.334173	4.384998
wind_speed_10m:ms	5.394532	3.948634
sfc_pressure:hPa	5.380634	3.827890
ceiling_height_agl:m	4.852037	3.967851
pressure_50m:hPa	4.646027	3.414725
wind_speed_v_10m:ms	4.761127	3.150840
is_in_shadow:idx	4.120510	3.720413
pressure_100m:hPa	3.349618	2.314721
t_1000hPa:K	3.326664	0.934157
air_density_2m:kgm3	2.662013	1.148611
fresh_snow_6h:cm	2.056228	1.544087
fresh_snow_12h:cm	2.070508	1.414530
estimated_diff_hours	1.864098	1.426947
snow_depth:cm	1.952748	1.137987
super_cooled_liquid_water:kgm2	1.831976	1.008976
fresh_snow_3h:cm	1.193375	0.742818
precip_5min:mm	1.352372	0.439489
dew_point_2m:K	1.169612	0.229684
dew_or_rime:idx	0.818131	0.387002
precip_type_5min:idx	0.757810	0.208112
fresh_snow_1h:cm	0.689184	0.252078
absolute_humidity_2m:gm3	0.409680	0.073343
rain_water:kgm2	0.326405	0.068361
prob_rime:p	0.265392	-0.060476
wind_speed_w_1000hPa:ms	0.000000	0.000000

```
snow_melt_10min:mm                0.043535   -0.233487
```

[ ]:
```
# feature importance
observed = train_data_with_dates[train_data_with_dates["ds"] < split_time]
observed = observed[observed["location"] == location]
predictors[0].feature_importance(feature_stage="original", data=observed,␣
↪time_limit=60*10)
```

```
These features in provided data are not utilized by the predictor and will be
ignored: ['ds', 'elevation:m', 'sample_weight', 'location', 'prediction']
Computing feature importance via permutation shuffling for 43 features using
5000 rows with 10 shuffle sets… Time limit: 600s…
        655.26s = Expected runtime (65.53s per shuffle set)
```

[ ]:
```
display(Javascript("IPython.notebook.save_checkpoint();"))
time.sleep(3)

subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
↪join('notebook_pdfs', f"{new_filename}_with_feature_importance.pdf"),␣
↪"autogluon_each_location.ipynb"])
```

[ ]:
```
# import subprocess

# def execute_git_command(directory, command):
#     """Execute a Git command in the specified directory."""
#     try:
#         result = subprocess.check_output(['git', '-C', directory] + command,␣
↪stderr=subprocess.STDOUT)
#         return result.decode('utf-8').strip(), True
#     except subprocess.CalledProcessError as e:
#         print(f"Git command failed with message: {e.output.decode('utf-8').
↪strip()}")
#         return e.output.decode('utf-8').strip(), False

# git_repo_path = "."

# execute_git_command(git_repo_path, ['config', 'user.email',␣
↪'henrikskog01@gmail.com'])
# execute_git_command(git_repo_path, ['config', 'user.name', hello if hello is␣
↪not None else 'Henrik eller Jørgen'])

# branch_name = new_filename

# # add datetime to branch name
# branch_name += f"_{pd.Timestamp.now().strftime('%Y-%m-%d_%H-%M-%S')}"

# commit_msg = "run result"
```

```
# execute_git_command(git_repo_path, ['checkout', '-b',branch_name])

# # Navigate to your repo and commit changes
# execute_git_command(git_repo_path, ['add', '.'])
# execute_git_command(git_repo_path, ['commit', '-m',commit_msg])

# # Push to remote
# output, success = execute_git_command(git_repo_path, ['push',␣
 ↪'origin',branch_name])

# # If the push fails, try setting an upstream branch and push again
# if not success and 'upstream' in output:
#     print("Attempting to set upstream and push again...")
#     execute_git_command(git_repo_path, ['push', '--set-upstream',␣
 ↪'origin',branch_name])
#     execute_git_command(git_repo_path, ['push', 'origin', 'henrik_branch'])

# execute_git_command(git_repo_path, ['checkout', 'main'])
```